MongoDB History & Complete Notes (Engineering Students)

This document provides MongoDB history, concepts, architecture, data model, features, CRUD operations, aggregation, CAP theorem, and real-world usage. Queries and code examples are shown in bordered boxes for neat readability.

■ 1) MongoDB History (Timeline)

MongoDB Origin

MongoDB was started in 2007 by a company named 10gen. The first major release (v1.0) came around 2009. Later 10gen was renamed as MongoDB Inc.

Why MongoDB became popular

It became popular because it supports flexible schema, high performance, replication and horizontal scaling for modern web applications.

License Change

MongoDB later changed license to SSPL in 2018.

■ 2) What is MongoDB?

Definition

MongoDB is a NoSQL, document-oriented DBMS. It stores data as documents (JSON-like structure) inside collections. It is designed for high performance, high availability and scalability.

■ 3) MongoDB Data Model

RDBMS vs MongoDB Mapping

Database → Database
Table → Collection
Row → Document
Column → Field (Key)

Sample Document

Example MongoDB document:

```
{
  _id: 101,
  name: "Ravi",
  age: 20,
  grade: "A",
  skills: ["C", "Python"]
}
```

■ 4) BSON (Binary JSON)

What is BSON?

MongoDB stores documents internally in BSON (Binary JSON). BSON stores data with datatype information and supports extra datatypes like ObjectId, Date, Binary, etc.

■ 5) Key Features of MongoDB

Schema-less database

Different documents in the same collection can have different fields and datatypes. This provides flexibility during development.

High Performance

MongoDB supports indexing, embedding documents and optimized storage which makes read/write operations faster.

High Availability (Replication)

MongoDB uses Replica Sets where Primary handles writes and Secondary nodes replicate data. If Primary fails, Secondary automatically becomes Primary.

Horizontal Scalability (Sharding)

MongoDB supports Sharding (partitioning). Large data is divided across multiple servers (shards) for better scalability.

■ 6) MongoDB Architecture

mongod

MongoDB server process which stores data and handles requests.

mongosh

MongoDB Shell / client used to interact with database.

MongoDB Compass

Official GUI tool for MongoDB, used to visually manage database.

■ 7) MongoDB Query Language (MQL)

What is MQL?

MongoDB uses MongoDB Query Language (MQL). It looks similar to JavaScript object syntax.

Example Query

Example: find students with age > 20

```
db.students.find({ age: { $gt: 20 } })
```

■ 8) CRUD Operations in MongoDB

Create (Insert)

Insert 1 document into collection:

```
db.students.insertOne({ name:"Sita", age:21 })
```

Read (Find)

Read documents matching condition:

```
db.students.find({ age:21 })
```

Update

Update one document using $set:

```
db.students.updateOne({ name:"Sita" }, { $set:{ age:22 } })
```

Delete

Delete one document:

```
db.students.deleteOne({ name:"Sita" })
```

■ 9) Aggregation Pipeline (Advanced)

Aggregation Usage

Aggregation is used for grouping, counting, sum, average, sorting etc.

Example Aggregation

Count students grade wise for Hyderabad city:

```
db.students.aggregate([
  { $match: { city: "Hyderabad" } },
  { $group: { _id: "$grade", total: { $sum: 1 } } }
])
```

Meaning

$match = filter like WHERE in SQL
$group = GROUP BY in SQL

■ 10) CAP Theorem & MongoDB

CAP Theorem

In distributed databases, we can guarantee only two among Consistency, Availability, Partition tolerance.

MongoDB

MongoDB generally aims for CP (Consistency + Partition tolerance) with replica sets and write concerns.

■ 11) Where MongoDB is used?

Applications

MongoDB is widely used in MERN full stack applications, social media apps, e-commerce, IoT data, analytics, logs, content management systems and real-time apps.

■ 12) Advantages and Disadvantages

Advantages

Flexible schema, fast development, stores complex structures like arrays & nested documents, scalable using sharding, good integration with JSON/Node.js apps.

Disadvantages

Not best for complex joins, schema-less may cause datatype mismatch, needs careful indexing, duplication may occur.

■ Final Exam 3-Line Summary

Summary

MongoDB is a NoSQL document-oriented DB started in 2007 by 10gen and released around 2009. It stores data as BSON documents in collections with flexible schema. It supports replication for availability and sharding for scalability.

MongoDB Installation & Usage in Windows (Student Notes)

This document explains step-by-step installation of MongoDB in Windows and how to use it using Compass, mongosh (shell), and Node.js.

■ A) How to Install MongoDB in Windows

Step 1: Download MongoDB Community Server

Download MongoDB Community Server (Windows MSI) from official MongoDB website.
Select OS = Windows, Package = MSI, Version = latest stable.

Step 2: Install using MSI Setup

During installation select:
1) Complete Setup
2) Install MongoDB as a Service (Recommended)
3) Install MongoDB Compass (GUI tool)

Step 3: Install MongoDB Shell (mongosh)

Install mongosh separately (MongoDB Shell MSI for Windows).

■ B) How to Start MongoDB in Windows

Method 1: Start MongoDB Service (Recommended)

```
net start MongoDB
```

Stop MongoDB Service

```
net stop MongoDB
```

Method 2: Start Manually (If service not installed)

1) Create the database folder:
C:\data\db

2) Run mongod server:

```
"C:\Program Files\MongoDB\Server\7.0\bin\mongod.exe" --dbpath="C:\data\db"
```

Expected Output

If it runs correctly you will see:
■ waiting for connections

■ C) How to Use MongoDB in Windows

Way 1: Using MongoDB Compass (GUI)

1) Open MongoDB Compass
2) Connect using: mongodb://localhost:27017

Compass allows: create DB, collections, insert data, run queries in GUI.

Way 2: Using mongosh (Terminal)

Open Command Prompt and type:

```
mongosh
```

Basic mongosh commands

Use the following commands:

```
show dbs
use college
db.students.insertOne({ name: "Ravi", age: 20, grade: "A" })
db.students.find().pretty()
```

Way 3: Use MongoDB in Node.js (Full Stack)

Install MongoDB driver:

```
npm install mongodb
```

Node.js MongoDB Connection Example

Example program:

```
const { MongoClient } = require("mongodb");

const url = "mongodb://localhost:27017";
const client = new MongoClient(url);

async function run() {
  await client.connect();
  const db = client.db("college");
  const students = db.collection("students");

  await students.insertOne({ name: "Sita", age: 21 });
  console.log("Inserted!");

  const data = await students.find().toArray();
  console.log(data);

  await client.close();
}

run();
```

■ Viva Important Points

MongoDB server program

mongod

MongoDB shell / client

mongosh

MongoDB GUI tool

MongoDB Compass

Default port number

27017

MongoDB Basic & Advanced Queries (With Examples)

This notes sheet covers CRUD, filters, array queries, embedded docs, aggregation, indexing and joins in MongoDB. All queries are written in a bordered code box for easy readability.

■ 1) BASIC QUERIES (CRUD)

1. Insert One Document

```
db.students.insertOne({ name: "Ravi", age: 20, grade: "A", city: "Hyderabad" })
```

Inserts 1 document into collection.

2. Insert Many Documents

```
db.students.insertMany([
  { name: "Sita", age: 21, grade: "B", city: "Warangal" },
  { name: "Kiran", age: 19, grade: "A", city: "Karimnagar" }
])
```

Inserts multiple documents at once.

3. Find All Documents

```
db.students.find()
```

Displays all documents.

4. Find Pretty Output

```
db.students.find().pretty()
```

Shows formatted output.

5. Find One Document

```
db.students.findOne({ name: "Ravi" })
```

Returns first matching document.

6. Projection (Select Fields)

```
db.students.find({ city: "Hyderabad" }, { name: 1, grade: 1, _id: 0 })
```

Shows only name and grade (hides _id).

7. Update One Document

```
db.students.updateOne(
  { name: "Ravi" },
  { $set: { grade: "A+", age: 21 } }
)
```

Updates first matching document.

8. Update Many Documents

```
db.students.updateMany(
  { city: "Hyderabad" },
  { $set: { state: "Telangana" } }
)
```

Updates all matching documents.

9. Replace One Document

```
db.students.replaceOne(
  { name: "Kiran" },
  { name: "Kiran", age: 20, grade: "B", city: "Nizamabad" }
)
```

Replaces entire document (old fields removed).

10. Delete One Document

```
db.students.deleteOne({ name: "Ravi" })
```

Deletes first matching document.

11. Delete Many Documents

```
db.students.deleteMany({ city: "Warangal" })
```

Deletes all matching documents.

■ 2) BASIC FILTER QUERIES

12. Equal Condition

```
db.students.find({ grade: "A" })
```

Returns students with grade A.

13. AND Condition

```
db.students.find({ city: "Hyderabad", grade: "A" })
```

Both conditions must be true.

14. OR Condition

```
db.students.find({ $or: [{ city: "Hyderabad" }, { city: "Warangal" }] })
```

Any one condition true.

15. NOT EQUAL ($ne)

```
db.students.find({ city: { $ne: "Hyderabad" } })
```

Students not from Hyderabad.

16. Greater / Less ($gt, $lt)

```
db.students.find({ age: { $gt: 20 } })
```

age > 20.

17. Range ($gte, $lte)

```
db.students.find({ age: { $gte: 20, $lte: 22 } })
```

Age between 20 and 22.

18. IN Query ($in)

```
db.students.find({ city: { $in: ["Hyderabad", "Warangal"] } })
```

City matches from list.

19. NOT IN ($nin)

```
db.students.find({ grade: { $nin: ["A", "B"] } })
```

Grade not A or B.

■ 3) SORT, LIMIT, SKIP

20. Sort Ascending

```
db.students.find().sort({ age: 1 })
```

Sort by age increasing.

21. Sort Descending

```
db.students.find().sort({ age: -1 })
```

Sort by age decreasing.

22. Limit

```
db.students.find().limit(3)
```

Displays only 3 documents.

23. Skip (Pagination)

```
db.students.find().skip(3).limit(3)
```

Skip first 3 show next 3.

■ 4) ADVANCED QUERIES

24. Count Documents

```
db.students.countDocuments()
```

Total count.

25. Count with Condition

```
db.students.countDocuments({ grade: "A" })
```

Count grade A students.

26. Distinct Values

```
db.students.distinct("city")
```

Gives unique city names.

27. Regex Search

```
db.students.find({ name: { $regex: "^R" } })
```

Names starting with R.

■ 5) ARRAY QUERIES

Insert Sample Document (Array)

```
db.students.insertOne({
  name: "Anil",
  skills: ["C", "Python", "MongoDB"],
  marks: [70, 85, 90]
})
```

Sample document with arrays.

28. Array contains value

```
db.students.find({ skills: "MongoDB" })
```

Finds document if array contains MongoDB.

29. Array All Values ($all)

```
db.students.find({ skills: { $all: ["C", "Python"] } })
```

Must contain both values.

30. Array Size ($size)

```
db.students.find({ skills: { $size: 3 } })
```

Array size is 3.

■ 6) EMBEDDED DOCUMENT QUERIES

Insert Sample Embedded Doc

```
db.students.insertOne({
  name: "Suresh",
  address: { city: "Hyderabad", pincode: 500001 }
})
```

Sample embedded document.

31. Query inside embedded doc

```
db.students.find({ "address.city": "Hyderabad" })
```

Dot notation used.

■ 7) ADVANCED UPDATES

32. Increment ($inc)

```
db.students.updateOne(
  { name: "Sita" },
  { $inc: { age: 1 } }
)
```

Increases age by 1.

33. Rename Field ($rename)

```
db.students.updateMany({}, { $rename: { city: "district" } })
```

Renames city → district.

34. Remove Field ($unset)

```
db.students.updateMany({}, { $unset: { district: "" } })
```

Deletes field.

35. Push into Array ($push)

```
db.students.updateOne(
  { name: "Anil" },
  { $push: { skills: "NodeJS" } }
)
```

Adds element to array.

36. Unique Array Insert ($addToSet)

```
db.students.updateOne(
  { name: "Anil" },
  { $addToSet: { skills: "Python" } }
)
```

Adds only if not already present.

37. Remove from Array ($pull)

```
db.students.updateOne(
  { name: "Anil" },
  { $pull: { skills: "C" } }
)
```

Removes C from array.

■ 8) AGGREGATION

38. $match (WHERE)

```
db.students.aggregate([{ $match: { city: "Hyderabad" } }])
```

Filters documents.

39. $group (GROUP BY)

```
db.students.aggregate([{ $group: { _id: "$grade", total: { $sum: 1 } } }])
```

Groups by grade and counts.

40. Average age ($avg)

```
db.students.aggregate([{ $group: { _id: null, avgAge: { $avg: "$age" } } }])
```

Calculates average age.

41. Max & Min ($max, $min)

```
db.students.aggregate([{ $group: { _id: null, maxAge: { $max: "$age" }, minAge: { $min: "$age
```

Finds max and min age.

42. Aggregation Sort

```
db.students.aggregate([{ $sort: { age: -1 } }])
```

Sorts results.

43. Aggregation Limit

```
db.students.aggregate([{ $sort: { age: -1 } }, { $limit: 5 }])
```

Top 5 highest age.

■ 9) INDEXING

44. Create Index

```
db.students.createIndex({ name: 1 })
```

Speeds up search queries.

45. View Indexes

```
db.students.getIndexes()
```

Displays all indexes.

■ 10) JOINS ($lookup)

46. Join students and courses

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "courseId",
      foreignField: "courseId",
      as: "courseDetails"
    }
  }
])
```

Works like SQL JOIN.

■ 11) ADMIN COMMANDS

47. Show all Databases

```
show dbs
```

Lists databases.

48. Use database

```
use college
```

Switches to database.

49. Show collections

```
show collections
```

Lists collections.

50. Drop collection

```
db.students.drop()
```

Deletes collection.

51. Drop database

```
db.dropDatabase()
```

Deletes database.